PB84-190529

Implementation of the USEPA (United States Environmental Protection Agency) Regional Oxidant Modeling System

(U.S.) Environmental Sciences Research Lab.
Research Triangle Park, NC

May 84

U.S. Department of Commerce
National Technical Information Service
NTIS

PB84-190529

EPA-600/D-84-116
May 1984

Implementation of the USEPA Regional Oxidant Modeling System

Joan H. Novak*
U.S. Environmental Protection Agency

Environmental Sciences Research Laboratory
U.S. Environmental Protection Agency
Research Triangle Park, North Carolina 27711

ENVIRONMENTAL SCIENCES RESEARCH LABORATORY
OFFICE OF RESEARCH AND DEVELOPMENT
U.S. ENVIRONMENTAL PROTECTION AGENCY
RESEARCH TRIANGLE PARK, NC 27711

* On assignment from the National Oceanic and Atmospheric Administration, U.S. Department of Commerce

REPRODUCED BY
NATIONAL TECHNICAL
INFORMATION SERVICE
U.S. DEPARTMENT OF COMMERCE
SPRINGFIELD, VA. 22161

TECHNICAL REPORT DATA *(Please read Instructions on the reverse before completing)*		
1. REPORT NO. EPA-600/D-84-116	2.	3. RECIPIENT'S ACCESSION NO. PB84 190529
4. TITLE AND SUBTITLE Implementation of the USEPA Regional Oxidant Modeling System		5. REPORT DATE May 1984 6. PERFORMING ORGANIZATION CODE
7. AUTHOR(S) Joan H. Novak		8. PERFORMING ORGANIZATION REPORT NO.
9. PERFORMING ORGANIZATION NAME AND ADDRESS (Same as Block 12)		10. PROGRAM ELEMENT NO. CDWA1A/02-1322 (FY-84) 11. CONTRACT/GRANT NO.
12. SPONSORING AGENCY NAME AND ADDRESS Environmental Sciences Research Laboratory--RTP, NC Office of Research and Development U.S. Environmental Protection Agency Research Triangle Park, NC 27711		13. TYPE OF REPORT AND PERIOD COVERED 14. SPONSORING AGENCY CODE EPA/600/09
15. SUPPLEMENTARY NOTES		

16. ABSTRACT

Model design, implementation and quality assurance procedures can have a significant impact on the effectiveness of long term utility of any modeling approach. The Regional Oxidant Modeling System (ROMS) is exceptionally complex because it treats all chemical and physical processes thought to affect ozone concentration on a regional scale. Thus, to effectively illustrate useful design and implementation techniques, this paper describes the general modeling framework which forms the basis of the ROMS. This framework is flexible enough to allow straightforward update or replacement of the chemical kinetics mechanism and/or any theoretical formulations of the physical processes. Use of the Jackson Structured Programming (JSP) method to implement this modeling framework has not only increased programmer productivity and quality of the resulting programs, but also has provided standardized program design, dynamic documentation, and easily maintainable and transportable code. A summary of the JSP method is presented to encourage modelers to pursue this technique in their own model development efforts. In addition, since data preparation is such an integral part of a successful modeling system, the ROMS processor network is described with emphasis on the internal quality control techniques.

17. KEY WORDS AND DOCUMENT ANALYSIS		
a. DESCRIPTORS	b. IDENTIFIERS/OPEN ENDED TERMS	c. COSATI Field/Group
18. DISTRIBUTION STATEMENT RELEASE TO PUBLIC	19. SECURITY CLASS *(This Report)* UNCLASSIFIED 20. SECURITY CLASS *(This page)* UNCLASSIFIED	21. NO. OF PAGES 18 22. PRICE

EPA Form 2220-1 (9-73)

NOTICE

This document has been reviewed in accordance with U.S. Environmental Protection Agency policy and approved for publication. Mention of trade names or commercial products does not constitute endorsement or recommendation for use.

Introduction

Atmospheric modelers typically focus their attention on the theoretical aspects of a model. However, the quality of the model implementation also has a direct impact on the success or failure of a given modeling approach. Consequently, this paper describes the structure of the Environmental Protection Agency's Regional Oxidant Modeling System (ROMS) and the techniques used to implement both the theoretical model formulation[2] and input processor network[3]. The design, implementation, and quality control procedures presented here are applicable to any model development effort. Thus the purpose of this paper is to make modelers aware of the benefits of a general modeling framework as well as provide specific examples of effective design and implementation methods.

As models become more complex the potential impact of these design and implementation procedures increases significantly. The ROMS is being developed to aid in the formulation of regional ozone control strategies because urban scale models can not adequately treat long range transport of ozone and its precursors. The ROMS differs from most urban and regional models in that its predictions rely more heavily on theoretical science. The empirical data which provide a basis for the treatment of turbulence in urban diffusion models are inadequate for the regional scale. And most regional Eulerian models are in fact, merely extensions of urban scale models[1]. However the ROMS, also a Eulerian approach, combines observed data and theoretical formulations of all governing chemical and physical processes known, or thought to affect air pollution concentrations in a regional (1000 km) domain. The model formulation explicitly treats nighttime and mesoscale processes which have been ignored by other regional modeling approaches. Consequently, the complexity of this formulation and the need to constantly improve these process descriptions had led to the development of a general modeling framework which can not only respond to the ozone problem, but can potentially be adapted for other regional air pollution problems.

Structure of the Regional Oxidant Modeling System

The key components of the modeling system are illustrated in Figure 1. The system requires six types of "raw" data inputs: air quality, meteorology, emissions, cloud cover, land use, and topography. These "raw" data inputs are manipulated by a network of processors which range in function from simple reformatting of emissions data to generating the complex families of wind fields which drive the atmospheric transport. The function of each processor is summarized in Table I. These processors are interconnected by their requirements for and production of data. And the ultimate product of the processor network is a collection of data files which can be categorized into two types: processor input files (PIF) and model input files (MIF). Processor input files contain partially processed data required as input to higher level processors. Model input files also provide input to a number of high level processors, however they primarily contain the parameter fields which are transformed into matrix elements and variables required by the model algorithms.

The processors are organized into six distinct levels as illustrated in Figure 2. Level 1 processors interface directly with the "raw" input data

Table I. Processor Function Summary

Processor Name	Function
P01	Prepares rawind data
P02	Determines initial, upper and lateral boundary conditions
P03	Prepares surface meteorology data
P04	Estimates surface roughness, Obukhov length, surface heat flux and friction velocity
P05	Processes cumulus cloud fraction data
P06	Computes averaged terrain elevations
P07	Computes terrain penetration fractions, simulates the depth of the daytime shear layer, and the depth and motion of the nighttime surface inversion layer
P08	Computes parameters specifying pollutant fluxes across top surfaces of layers 2 and 3.
P09	Calculates adjustment factors for variation of chemical rate constants due to atmospheric density, temperature, and cloud cover
P10	Computes emissions source strength functions and plume volume fraction parameter
P11	Computes families of vertically integrated horizontal winds
P12	Computes parameters to describe interfacial material fluxes, and estimates horizontal eddy diffusivities
P13	Estimates line source length in each grid
P14	Prepares emissions inventory data
P15	Computes deposition resistance and deposition velocity
P16	Interpolates rawinds to hourly upper air profiles
P17	Computes time derivatives of cell heights
P19	Computes fractional sky coverage for all cloud types
P25	Prepares land use fractions in each grid
P29	Computes diffusivities and back tracked cell locations

sets. These "raw" data sets have, in fact, undergone extensive quality control. Processors at any level can interface directly with the B-Matrix compiler (BMC) described below, by production of model input files. This multi-level organization is important to the network because it clearly delineates the sequence of program execution. Processors at the same level may execute simultaneously. However, a processor at any given level must wait until all processors along its input data paths have been completed. Formal definition of all data/processor relationships and automation of processor executions are essential to ensure consistency and validity of model input files.

The program which serves as the interface between the model input files and the algorithms describing the governing processes is called the B-Matrix Compiler (BMC) because it functions similar to a computer language compiler which transforms high level language commands into a machine/algorithm specific representation. The BMC mathematically combines physical parameters such as wind components, horizontal eddy diffusivities, etc. into the complex coefficients required for solution of the governing equations. These coefficients can no longer be equated with physical reality; they are purely mathematical entities related specifically to the form of the generalized finite difference algorithms.

The core of the system is a set of algorithms which solve the coupled set of generalized finite difference equations that describe the governing processes in each layer of the model. These governing equations have been expressed in a form which allows the chemical kinetics, advection and vertical flux to be treated independently. Thus, for example, the chemical mechanism could be replaced without reworking other parts of the model. This chemistry module exchanges information with algorithms of the governing equations via two vectors. One vector contains the net production rate of each species and the other vector contains the net destruction rate of each species. Such design simplifications enhance the flexibility of the model and have been extended to allow interchange not only of the chemistry mechanism, but of all theoretical formulations of the physical and meteorological processes, i.e., the processors.

Thus the structure of the ROMS provides a flexible modeling framework with both permanent and interchangeable components. The permanent components are:

- the algorithmic form of the governing equations
- the B-Matrix Compiler
- the processor and model input files
- the data interfaces between components and processors

Discussions in a later section illustrate how careful file design can minimize the impact of unanticipated modifications on the "fixed" data interfaces. The interchangeable components are the chemistry kinetics mechanism and the processors.

The formal specification documents for the Regional Oxidant Modeling System, references 2 and 3, intentionally separate functional components and specifically define "fixed" data interfaces. This modular structure provides several benefits:

- ability to easily modify process descriptions and chemical kinetics mechanism
- ability to modify the general framework to handle other regional scale problems - visibility, particulate, acid deposition
- support of structured design techniques and team implementation
- simplification of testing, debugging and error isolation

Once such a general modeling framework is operational, continuous upgrading to state-of-the-art methodologies is not only feasible, but most practical. The long term usefulness for atmospheric modeling research and applications tremendously outweights any additional efforts required to develop a flexible framework rather than a specific model. After all, the difference in effort is primarily in the design, not in the implementation.

Design Method

The concept of functional modularity, however, does not in itself guarantee the benefits listed above. To actually provide the full scope of flexibility implied by these benefits, the system design method must do much more than merely implement a well structured formulation. What are the qualities of structured design and structured programming techniques that can ensure the long term usefulness and quality of a general modeling framework?

Experience has shown that at least the following products should necessarily be provided by the design technique:

- clear description of overall organization
- understandable dynamic documentation
- maintainable programs
- standardization of software design

Jackson Structured Programming

The Jackson Structured Programming (JSP) method[4] provides these essentials and has been used for the past few years in development of the Regional Oxidant Modeling System. The JSP method is typically not used for mathematical programming because it does not attempt to explicitly handle the development of abstract mathematical algorithms. However, development of a complex modeling system involves both data processing and mathematical aspects. A brief summary of the JSP method as applied in the ROMS development

will help clarify how both aspects of the system can be integrated to attain the primary goal of a flexible modeling framework.

The JSP method, as presented in reference 4, primarily focuses on the design of a single program. We have decomposed the development of the modeling system into independent functional programs linked by their data interfaces. The JSP method is used to design and implement each program component. The basic principle of the JSP method is that the program should be based on the fundamental structure of the problem. The method outlines how inputs, processes, and outputs are combined to form an adequate representation of the problem structure. This "model" of the problem structure leads directly to the program solution. The following five steps outline the JSP method:

Step 1: Review the system requirements as a whole and define, by appropriate diagrams, all input and output data structures. The JSP data structures are related to the context of a particular problem.

Step 2: Combine the input and output data structures associated with each component to form the program structure. Difficulty in merging data structures indicates potential inadequacy of data structure definitions or the existence of structure clashes. JSP provides several techniques for handling different types of structure clashes.

Step 3: List operations to be performed by the program and allocate to appropriate position on the program structure diagram. Difficulty here may indicate inadequate program structure or recognition problems. JSP also provides specific techniques to solve these recognition difficulties.

Step 4: Convert program structure to textual form and add conditions to iterations.

Step 5: Transform textual form to a specific programming language.

The first four steps of the JSP method are independent of hardware and programming language considerations. However, since the textual form is so close to a programming language implementation we chose to deviate slightly from the JSP method and move directly to FORTRAN code at the fourth step.

According to Webster's Dictionary a method implies a regular, orderly, logical procedure for doing something. JSP provides just that for program design. The starting point is well defined and when a choice is involved the method inherently recognizes whether the decision was good or bad, and at least rejects the bad choice. The consequence of a given choice can be evaluated as part of the design process, thus avoiding costly design changes discovered during implementation. Finally, the ability to complete the design by mechanistic treatment of difficulties and by allocation of operations and logic provides some assurance that the underlying structure is correct.

Consequently, the JSP treatment of data processing aspects of the system design is clearly defined. And similarly, complex mathematical algorithms are handled as operations performed on the underlying data structure. The

most difficult task here is to determine the appropriate data streams upon whose structure the algorithm is constructed. The correct program design ensures that all required data are provided to the algorithm at appropriate times and that the required form of the output data are available for input to subsequent programs.

The JSP method is fundamentally different from the popular top down approach. JSP merges data structures and operations to form a "model" of the problem structure which then transforms directly into the program solution. In contrast, the top down approach[5] decomposes the problem into segments, produces a tentative design based on the initial data flows, and iteratively refines the design during the implementation phase. The latter approach does not provide feedback as to the adequacy of segmentation and design decisions. The analyst must rely on experience and intuition for decision making where initial decisions are critical because they set the framework for all subsequent decisions. Inappropriate problem decomposition will eventually lead to system integration and testing problems which, at that late stage, may cause considerable coding changes throughout the system.

The top down approach is effective when applied to problems with known solutions. However, in a research environment where even the problems sometimes are not well defined, the JSP method has proven to be most effective.

The tools used in the JSP design process - initial requirements review, data and program structure diagrams, operations list, etc. - provide both overview and detailed documentation. The resulting software is maintainable because the steps of the design process can be repeated with system modifications in mind to isolate affected structures. Team implementation is more successful because analysts/programmers following the JSP method use standard decision criteria based on common data structures. The structure diagrams serve as a means of communicating both file organization and program design concepts. Moreover, use of the JSP method in the ROMS development has increased both programmer productivity and the overall quality of the software system.

Data Structure Implementation

Since data structure is a key element of the JSP method, it is useful to present further details of the data structures used in the ROMS. The system as a whole produces approximately one hundred data files, many of which have a common structure. The "raw" data files are actually outputs from data extraction and quality control systems which process the original form of the data. Typically each "raw" data file has a unique data structure. Fortunately, all other processor and model input files can be separated into five general types:

1) rawind station vertical profiles,

2) rawind station single level parameters,

3) surface meteorology and air quality parameters,

4) gridded time invariant-data, and

5) gridded time variant-data.

Subroutines were developed to read from and write to files of these five types. All processor and model input files are FORTRAN direct access and are identified by internal names of the form "PFnnn" or "MFnnn" where nnn can range from 000 to 999. PF indicates a processor input file, whereas MF indicates a model input file.

The first record of each file identifies the file by internal name, dates and times of creation and last update, structural definition of the file, description and units of parameters, grid definition, and time span of data in the file. The files are self documenting. The possibility of accessing the incorrect file is minimized when appropriate checks are performed by the software. Files are always accessed using their internal name so that data flow through the system can be consistently documented and easily traced. Unit numbers required by FORTRAN are dynamically assigned and attached by the processor I/O routines during execution. Files are also tagged by an external name associated with the physical disk file. The external name also designates a version of the file to permit identification of different versions of the same file structure.

The placement of the structural definition of the file in the header record provides the added measure of flexibility alluded to in a previous section. The structure of the file is defined by the following information: file type, length of header record, logical record length, length of block, maximum number of records, and number of parameters in the file. Even though the data interfaces between processors are theoretically unchangeable, experience has already shown that revised process descriptions may require a modification or addition of parameters in a given input file. This type of change is easily accommodated because of the dynamic nature of the file structure. In contrast, the addition of a single parameter to a rigid file structure might require extensive program modifications especially in view of the complexity of the data paths.

Modeling System Implementation

The system design has been implemented by a coordinated programmer team typically consisting of five members, although additional programming support has been used at various stages. Communication is maintained through formal and informal documentation, technical and design review meetings, and informal discussions. Table II lists the types of documentation that have been generated thus far in development of the Regional Oxidant Modeling System. The formal documents were created at the initial stages of development. Informal documentation has been created as part of the design process and will eventually be integrated to produce the formal operations and maintenance manuals. Automated cross reference listings are generated by the computer and have been updated continuously during system development. Truly dynamic working documentation is essential in the implementation of a system that is constantly evolving.

Table II. Documentation Summary

Formal

A Regional Scale (1000 km) Model of Photochemical Air Pollution

Part I. Theoretical Formulation

Part II. Input Processor Network Design

Informal

System Design

Processor and Data Structure Diagrams

Data Dictionaries

Processor Hierarchy Descriptions

Diagrams of Parameter Flow Through the System

Guidelines for Standardization of Coding Procedures

Automated Cross References:

processor - input/output files

processor - function

data file - parameters/units

data file - processor writing file/processor(s) reading file

Hardware Independence

The system is based on a computer independent design. And a prime goal of the implementation is to preserve the transportability of the code. Development of the system began on EPA's UNIVAC 1100 mainframe computer and was shifted to a VAX 11/780 minicomputer in May 1983. The model core has been installed on a CRAY-1 computer at Los Alamos National Laboratory in preparation for eventual applications. Strict adherence to the American National Standards Institute (ANSI) FORTRAN 77 full language specification and the use of inversion techniques[4,5] for I/O implementation have eliminated most of the transportability problems.

Processor Network Operation

Once the components of the modeling system are implemented, serious thought must be given to how the system operates as a whole during the testing, validation, and application/research phases. The model execution is isolated from the rest of the system by the nature of the singular data interface, the BMC, to the processor network. Each unique execution of the model requires a different file of matrix inputs. A complication arises, however, in the generation of the model input files required by the BMC. For example, the model typically would require a matrix input file for each wind field of interest. The wind fields are generated in P11 which in turn provides inputs to P29 and P10 (See Figure 2 for hierarchical relationships). Thus P11, P29, and P10 must be re-executed and their new output merged with all other model input files. This by itself is not overly complicated. However, envision the sequence of events that must take place when updated surface meteorological data (P3) are required; all high level processors must be re-executed. Any specific update, whether data or processor, requires a distinct sequence of processor executions. In a research environment where multiple investigations are often proceeding simultaneously, the most effective way to avoid confusion and preserve the quality of the research is to automate processor and BMC executions. Consequently, an "operating system" is being designed to perform the following functions:

- o Automatically produce runstreams for execution of all processors required to accomplish a specified task
  - Allow choice of processor version at any step
  - Allow choice of data file version at any step
- o Automatically produce runstream for execution of BMC
  - Allow specific selection of MIF versions
- o Provide data base structures for complete audit trail

The automation of runstream generation relies on the existence of software and data libraries conforming to pre-specified naming conventions. These standard conventions were created as part of the initial design. The libraries, however, must be carefully controlled to ensure the integrity of both software

and data. Thus, certification procedures have been developed to provide for necessary management of the libraries.

Certification Procedures

Software certification procedures are performed on all software components, typically programs, when development is thought to be complete. The process ensures that all required software, documentation, data files and control streams have been provided. The specification document and code are reviewed for accuracy and completeness, and are compared to locate any discrepancies. The code is recompiled, linked and executed with representative test data files provided. Test procedures, manual checks, and documentation are reviewed for accuracy and completeness. The certification process is not complete until all discrepancies have been resolved. At that point the software, test data files and control streams are transferred to certified libraries where write access is strictly controlled. Data files generated by certified software are also maintained in certified controlled access libraries provided that all inputs to the software were certified data files. A record is kept of all changes to certified components, including what changes were made, by whom, when and why. Modified components must be recertified before being placed in the certified libraries. Related uncertified software and data files are also collected in "test" libraries and are therefore available for use by the "operating system" when a particular version is requested.

Execution Audit Trails

The certification process provides a method of software change control and version identification. The processor "operating" system automates the production of desired data files. Yet, how can a program output file or listing confirm what actual inputs or program versions were used for a specific execution? The final technique that completes the quality control process is the execution audit trail. Fundamentally, the audit trail is nothing more than a printed summary identifying 1) all input data files by key information stored on their header records, and 2) all important software components by version number. The output file is keyed to the listing by the update date/time in its header record. The data file audit trail is implemented by including in each program specific code which opens each input file, reads the header record and prints the required summary. The software audit trail is implemented by inserting in each important software component - program and/or subroutine - a unique entry point which does nothing more than print the version identification of the component. Thus at the beginning of each program execution all "special" entry points are called to print a summary of software identification. These techniques are enforced through the certification process.

Conclusions

A description of the implementation of the Regional Oxidant Modeling System has been given to illustrate several basic concepts of importance to model developers:

1. The ultimate utility of a well designed and carefully implemented general framework for modeling is tremendous. The benefits of this approach warrent careful consideration when new model development efforts are initiated.

2. The Jackson Structured Programming method has proven to be extremely effective in design and implementation of the Regional Oxidant Modeling System. I hope the brief introduction to JSP presented here will encourage modelers to pursue this technique.

3. Effective quality control is likely to become an integral part of a software system. Both software transportability and the quality of the documentation affect the long term usefulness of a software system.

Acknowledgement

A large segment of the design and implementation work described was performed by Computer Data System, Incorporated under a General Service Administration interagency agreement DW930026-01-1.

References

1. R. G. Henderson, R. P. Pitter and J. Wisniewski, "Research Guidelines for Regional Modeling of Fine Particulates, Acid Deposition and Visibility", Report of a Workshop held at Port Deposit, MD October 29 - November 1, 1979.

2. R. G. Lamb, "A Regional Scale (1000 km) Model of Photochemical Air Pollution. Part 1: Theoretical Formulation." USEPA EPA-600/3-83-035, Research Triangle Park, NC, September 1982.

3. R. G. Lamb, "A Regional Scale (1000 km) Model of Photochemical Air Pollution. Part 2: Input Processor Network Design." USEPA publication pending, Research Triangle Park, NC, November 1983.

4. M. A. Jackson, Principles of Program Design, Academic Press, London, 1975.

5. J. R. Cameron, JSP & JSD: The Jackson Approach to Software Development, IEEE Computer Society Press, Maryland, 1983. pp. 5-7.

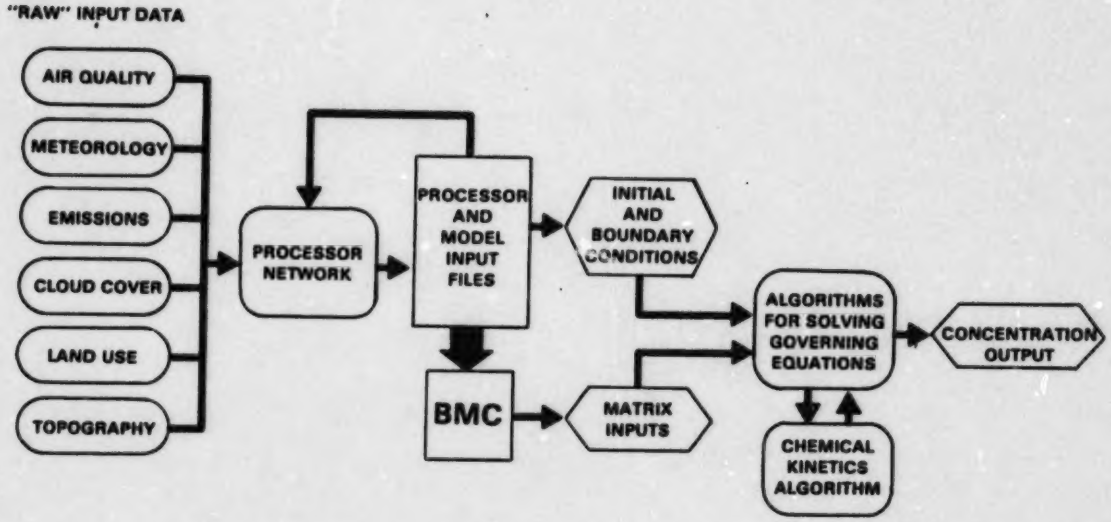

Figure 1. Regional Oxidant Modeling System Overview

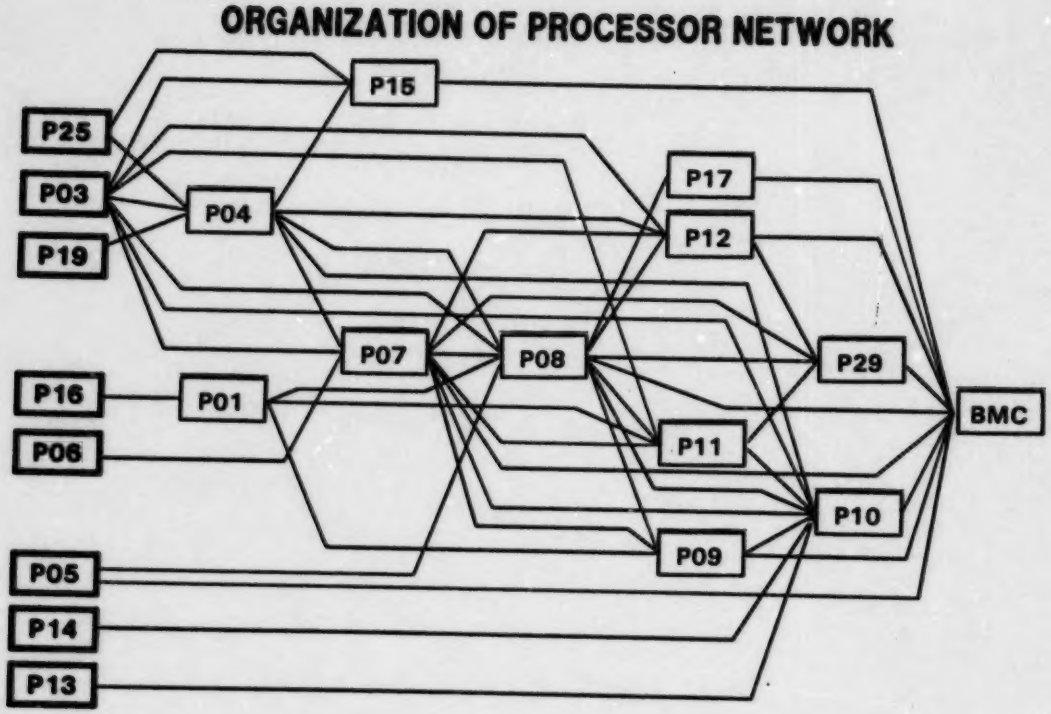

Figure 2. Organization of Processor Network

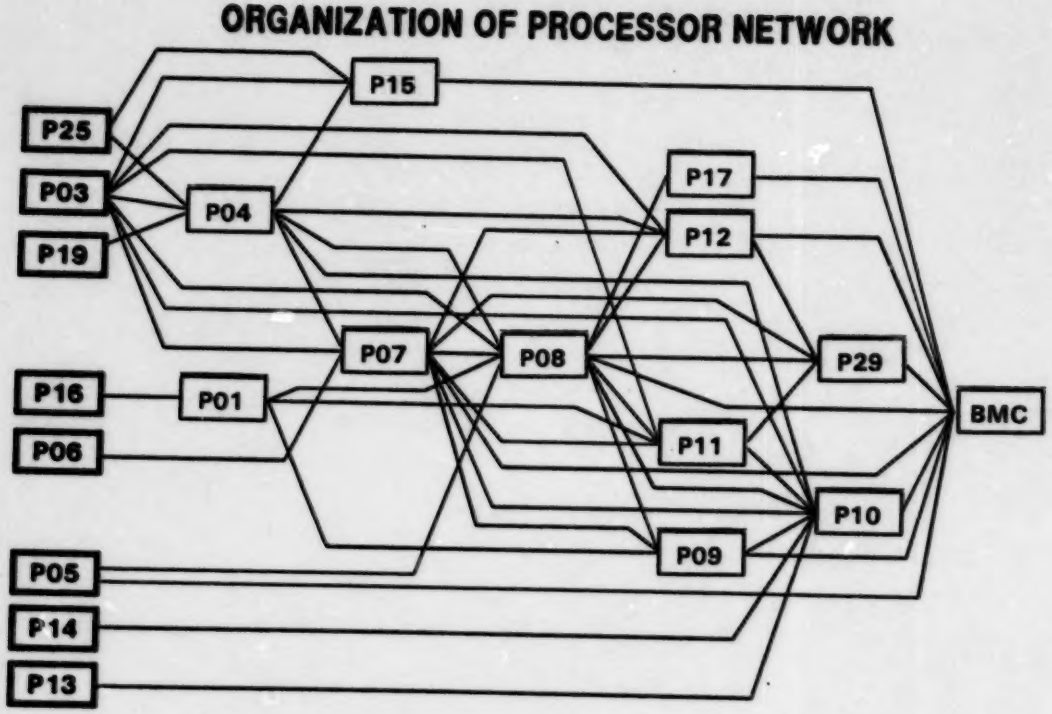

Figure 2. Organization of Processor Network